**Question 1:**

The first question asked us to generate {wi, xi} weighted pairs using 6 different N-values (17, 33, 65, 129, 257, 513) for all 3 Newton-Cotes methods:

1. Composite Midpoint rule (QuadSchemeMidpoint.m)
2. Composite Trapezoid rule (QuadSchemeTrapezoid.m)
3. Composite Simpson formula (QuadSchemeSimpson.m)

Implementation of the functions to get wi, xi values were more-or-less the same for all three methods. Vectors of size-N were pre-generated to hold the wi, xi, then Δx was calculated. Finally, a loop was used to handle generating {wi, xi } pairs with respect to N-number of partitions. Edge cases were handled outside the loop. The equations that were used to calculate the values for said pairings were obtained from the Numerical Integration slides presented in class.

This question did not ask for results and figures to be generated.

**Question 2:**

The Gaussian-Legendre was created via mapping the {wi, xi} pairs that were generated for [-1 1] to [a b]. The pairings that were produced were hard-coded and the value of N determined which sets of pairing was used via a switch statement. Then, like the methods in Question 1, the formula for Gauss was borrowed from the slides and implemented.

As with question 1, this question did not ask for results and figures to be generated.

**Question 3:**

Many methods were at work in Question 3, as the question itself had three major components. To create the requested convergence plots for the 3 Newton-Cotes methods for N = 17, 33, 65, 129, 257, 513 (representing the number of subintervals) and determine which method converged the quickest as N grew. The N-values 1-dimension vectors were placed in a vector, and other 1-dimensional vectors were initialized with zeroes to hold the composite integration results and subsequent error calculations. The function f(x) given in the problem was passed into a function handler, and the range of a and b were hard-coded.

A for-loop was used to iterate each of the N values in the vector. During each loop, {wi, xi} pairs were calculated using the three methods implemented in the first question (Composite Midpoint Rule, Composite Trapezoid Rule, and Composite Simpson’s Formula), thereby yielding three sets of {wi, xi} pairs.

All three of the methods share a common integral summation, ∑wif(xi), so I made a helper function called weightPairSum to assist with the ‘piecewise’ integral summation. This function essentially took the {wi, xi} pairs for each of the three methods and the f(x) function, returning the composite integration calculation for each respective method.

To determine how accurate each Newton-Cotes integration method was, they needed to be compared to the exact integration of f(x). Thus, the exact integral of f(x) over [a b] was calculated and placed in an N-spaced vector; N does not affect the exact integral value, but we need all vectors to span over the same N for plotting, thus the same integral value was duplicated number of N-times to fit a vector of appropriate length. Errors for each Newton-Cotes was calculated by subtracting the integration obtained by the method from the true integral value for every N-value.

Six figures were plotted: each of the three Newton-Cotes methods were plotted against the true integration as well as their respective errors. The error plots were easier to distinguish how quickly each method converged to zero; the closer the error was to zero, the more accurate the composite integration method was.

**Figure 1:** Composite Midpoint Rule Error Convergence

\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Figure 2:** Composite Trapezoid Rule Error Convergence



\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Figure 3:** Composite Simpson Formula Error Convergence



\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Table 1:** Error Values for Composite Newton-Cotes methods with Respect to N

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Midpoint Error** | **Trapezoid Error** | **Simpson’s Error** |
| 17 | -0.000390633268067 | 0.000573053429959 | -0.000114515002546 |
| 33 | -2.34078647070e-05 | 3.01931351884e-05 | -6.73544644502e-06 |
| 65 | -1.49151675898e-06 | 1.81188560955e-06 | -4.26933215003e-07 |
| 129 | -9.51189536025e-08 | 1.12110846828e-07 | -2.71895927995e-08 |
| 257 | -6.02159477836e-09 | 6.98940283428e-09 | -1.72065917070e-09 |
| 513 | -3.79023035180e-10 | 4.36558345029e-10 | -1.08295594714e-10 |

As demonstrated visually by Figures 1 to 3 (the error plots) above and confirmed numerically by Table 1, Simpson’s formula converged the fastest of the three Newton-Cotes methods. This is probably due to the Trapezoidal rule being an average of the left and right Riemann Sums; as a result, the Composite Trapezoid Rule performs poorly when it comes to approximating very curvy equations, like ours which involves a few sine and cosines. The same idea occurs with the Composite Midpoint Rule as it samples the midpoint between two subintervals and uses that lone point as the basis of interpolation.

Simpson’s Rule, however, places a parabola across every two subintervals on top of the fact that it also utilizes at least twice the number of subintervals compared to the other two. Generally, using more subintervals to approximate integrals yields a more accurate result, and such is the case we see here.

The next component of Question 3 was to estimate the explicit errors for the Trapezoid Rule and Simpson’s Formula, which was done through estimating the appropriate derivatives.

To estimate the derivative f’’(ζ) for the Composite Trapezoid Rule, for each N-value, the second derivative of f(x) was obtained by passing f(x) into the diff() routine and setting the nth derivative to 2. Then f’’(x) was calculated across the domain of a to b; the maximum f’’(x) was obtained, and it’s respective x value was verified to fall between a and b. This maximum f’’(x) value thus became the derivative estimation f’’(ζ) for the Composite Trapezoid Rule. Obtaining f(4)(φ), the derivative estimation for Composite Simpson’s, used the exact same methodology, except we were operating with the 4th, not the 2nd, derivative of f(x).

In our case, the derivative estimations results were:

f’’(ζ) = f2\_eta = 8.6612

f(4)(φ) = f4\_eta = 384.7914

Δx values were calculated for both Composite Simpson’s and Composite Trapezoid. The derivative estimations and Δx were then plugged into the explicit error formulas for each respective method, and the explicit error obtained.

Calculating the errors via Richardson’s Extrapolation was an extension of what has already been calculated. Due to Richardson’s Extrapolation using both Δx and Δx/2, we needed to calculate the composite integration of 2\*N using the Simpson’s and Trapezoid, as a half-step in x is equivalent to doubling the number of subintervals; we already calculated the composite integration of N in the previous component of this question. Obtaining the 2\*N integration used the same methods in the previous component: calculating the appropriate {wi, xi} pairs using the functions developed in Question 1, then feeding the generated {wi, xi} pairs into the weightPairSum function.

Calculating Richardson’s Extrapolation error was only a matter of plugging the integral calculation using N and integral calculation of 2\*N for a Newton-Cotes method into the error formula, which I’ve hard coded into a separate function called RichardsonExtrap().

**Table 2:** Error Values for Composite Trapezoid and Composite Simpson’s Formula methods using both Explicit Error and Richardson’s Extrapolation Error Formulas with Respect to N

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Explicit Error** | | **Richardson’s Extrapolation** | |
| **N** | **Trapezoid Rule** | **Simpson’s Formula** | **Trapezoid Rule** | **Simpson’s Formula** |
| 17 | 0.6993538615661 | 0.0009790763802 | 0.0007285920531 | -0.000144746249 |
| 33 | 0.1748384653915 | 6.895352924e-05 | 3.798786843e-05 | -8.44533574e-06 |
| 65 | 0.0437096163478 | 4.580984607e-06 | 2.270955285e-06 | -5.34096383e-07 |
| 129 | 0.0109274040870 | 2.952931808e-07 | 1.403061892e-07 | -3.39939620e-08 |
| 257 | 0.0027318510217 | 1.874475488e-08 | 8.741642669e-09 | -2.15093721e-09 |
| 513 | 0.0006829627554 | 1.180708796e-09 | 5.458543247e-10 | -1.35366681e-10 |

Figure X:

